Proceso de Software

Modelos de Ciclo de Vida

1. MODELO CASCADA

1.1. MODELO CASCADA BÁSICO O MODELO LINEAL SECUENCIAL

Uno de los primeros modelos que han sido propuestos es el modelo en cascada, ilustrado en la Figura 1, donde las etapas se representan cayendo en cascada, desde una etapa hacia la siguiente (Royce 1970). Como se desprende de esta figura, una etapa de desarrollo debe completarse antes de dar comienzo a la siguiente. De esta forma, cuando todos los requerimientos del cliente han sido identificados, analizados para comprobar su integridad y consistencia, y documentados en un documento de requerimientos, recién entonces el equipo de desarrollo puede seguir con las actividades de diseño del sistema. El modelo en cascada presenta una visión muy clara de cómo se suceden las etapas durante el desarrollo, y sugiere a los desarrolladores cuál es la secuencia de eventos que podrán encontrar.

El modelo en cascada ha sido utilizado para prescribir las actividades de desarrollo de software en una variedad de contextos. Por ejemplo, por muchos años ha sido la base para las entregas del desarrollo del software para el Departamento de Defensa de los EE.UU. y se encuentra definido en la norma Estándar 2167-A del mencionado organismo. Asociados con cada actividad del proceso estaban los hitos y las entregas de modo que los gerentes de proyecto podían usar el modelo para calibrar cuán cerca de la realización se encontraba un proyecto en un momento determinado. Por ejemplo, la etapa "prueba unitaria y de integración" en el modelo en cascada termina con el hito "módulos escritos, probados e integrados"; el producto intermedio es una copia del código probado. Después, el código puede derivarse a los verificadores de sistemas para que pueda ser combinado con otros componentes del sistema (hardware o software) y probado como un todo mayor.

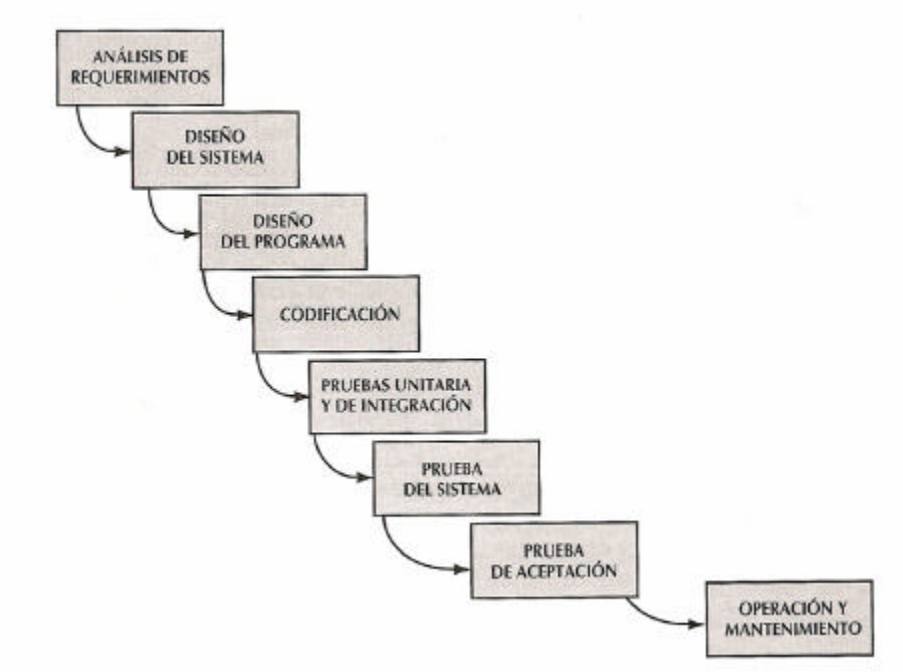

Figura 1 – Modelo Cascada

Etapas

1. **Ingeniería y modelado de Sistemas/Información.** Como el software siempre forma parte de un sistema más grande (o empresa), el trabajo comienza estableciendo requisitos de todos los elementos del sistema y asignando al software algún subgrupo de estos requisitos. Esta visión del sistema es esencial cuando el software se debe interconectar con otros elementos como hardware, personas, repositorios. La ingeniería y el análisis de sistemas acompaña a los requisitos que se recogen en el nivel de sistema con una pequeña parte de análisis y de diseño. La ingeniería de la información acompaña a los requisitos que se recogen en el nivel estratégico de empresa y en el nivel del área de negocio.
2. **[Requisitos]Análisis de los requisitos del software.** El proceso de reunión de requisitos se intensifica y se centra especialmente en el software. Para comprender la naturaleza de el (los) programa(s) a construirse, el ingeniero del software debe comprender el dominio de información del software, así como la función requerida, comportamiento, rendimiento, e interconexión. El cliente documenta y repasa los requisitos del sistema y del software.
3. **Diseño.** El diseño del software es realmente un proceso de muchos pasos que se centra en cuatro atributos distintos de un programa: estructura de datos, arquitectura del software, representaciones de interfaz y detalle procedimental (algoritmo). El proceso de diseño traduce requisitos en una representación del software que se pueda evaluar por calidad antes de que comience la generación de código. Al igual que los requisitos, el diseño se documenta y se hace parte de la configuración del software.
4. **Codificación.** Si se lleva a cabo un diseño en forma detallada, la generación de código se realiza mecánicamente.
5. **[Prueba]Pruebas de Integración.** Una vez que se ha generado el código, comienzan las pruebas del (los) programa(s) afectados. Se debe verificar que el software cumple con los requisitos definidos, y validar la correcta funcionalidad del producto.
6. **[Operación]Mantenimiento.** El software sufrirá cambios después de ser entregado al cliente. Dichos cambios pueden ocurrir porque se han detectado errores, porque el software debe adaptase para acoplarse a los cambios de su entorno externo, o porque el cliente desea mejoras funcionales o de rendimiento. El mantenimiento vuelva a aplicar cada una de las fases precedentes a un programa ya existente.

Una variación del modelo en cascada permite volver atrás en las actividades. Este modelo permite iteraciones durante el desarrollo, ya sea dentro de un mismo estado, como también desde un estado hacia otro anterior. La mayor iteración se produce cuando una vez terminado el desarrollo y cuando se ha visto el software producido, se decide comenzar de nuevo y redefinir los requisitos del usuario.

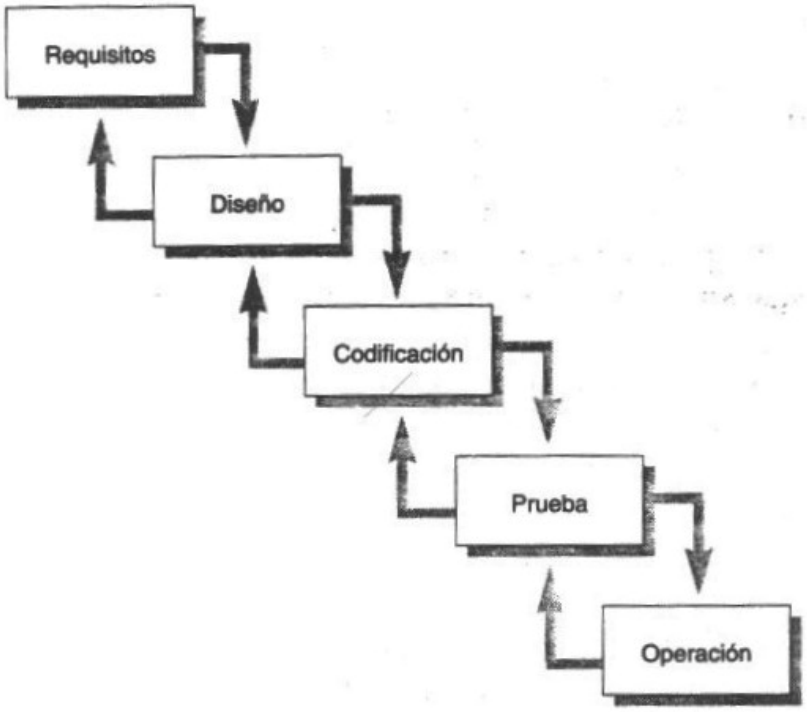

Figura 2 – Modelo Cascada con Iteraciones

Limitaciones para la utilización de este modelo:

- Los proyectos reales raras veces siguen el modelo secuencial que propone el modelo. Aunque el modelo lineal puede acoplar interacción, lo hace indirectamente. A menudo, durante el desarrollo, se pueden tomar decisiones que den lugar a diferentes alternativas. El modelo en cascada no reconoce esta situación. Por ejemplo, dependiendo del análisis de requisitos se puede implementar el sistema desde cero, o adoptar uno existente, o comprar un paquete que proporcione las funcionalidades requeridas.
- El modelo asume que los requisitos son definidos en su totalidad antes de comenzar el diseño, y luego no tienen modificaciones. A menudo es difícil que el cliente exponga explícitamente todos los requisitos, y mucho más que no los cambie con el paso del tiempo.
- Congelar los requisitos a su vez requiere seleccionar el hardware. La terminación de un gran proyecto puede llevar años. Dada la velocidad de obsolescencia de la tecnología es bastante probable que el software final utilice un hardware obsoleto.
- Una versión de trabajo del (los) programa(s) no estará disponible hasta que el proyecto esté muy avanzado. La mayor parte del feedback del cliente sobre sus necesidades se obtiene una vez que se han consumidos los recursos. Un grave error puede ser desastroso si no se detecta hasta que se revisa el programa.
- Los responsables del desarrollo del software siempre se retrasa innecesariamente. En un interesante análisis de proyectos reales, Bradac dijo que la naturaleza lineal del ciclo de vida clásico lleva a "estados de bloqueo" en el que algunos miembros del equipo del proyecto deben esperar a otros miembros del equipo para completar tareas dependientes. En efecto, el tiempo que se pasa esperando puede sobrepasar el

tiempo que se emplea en el trabajo productivo. Los estados de bloqueo tienden a ser más importantes al principio y al final de un proceso lineal secuencial.

- El modelo en cascada muestra cómo cada fase mayor del desarrollo concluye en la producción de algún artefacto (como los requerimientos, el diseño o el código). No revela la forma en que cada actividad transforma un artefacto en otro, tal como el paso de los requerimientos al diseño. Por ende, el modelo no proporciona guías a los gerentes y desarrolladores acerca de cómo manejar los cambios a los productos y las actividades que probablemente ocurran durante el desarrollo. Por ejemplo, cuando los requerimientos cambian durante las actividades de codificación, los cambios consiguientes del diseño y del código (que efectivamente se producen) no son tratados por el modelo en cascada. [McCracken y Jackson (1981)]
- La limitación principal del modelo en cascada reside en que no trata al software como un proceso de resolución de problemas. El modelo en cascada deriva del mundo del hardware y presenta una visión de manufactura sobre el desarrollo del software. Pero la manufactura produce un artículo en particular y lo reproduce muchas veces. El software no se desarrolla de la misma manera; en cambio, evoluciona a medida que el problema se comprende y se evalúan las alternativas. Así el software es un proceso de creación, no de fabricación. El modelo en cascada nada dice de las típicas actividades de avance y retroceso que llevan a la creación del producto final. En particular, normalmente la creación implica intentar un poco de esto y de aquello, como desarrollar y evaluar prototipos, valorizar la factibilidad de los requerimientos, comparar varios diseños, aprender a partir de los errores y, eventualmente, establecer una solución satisfactoria al problema en cuestión.[Curtis, Krasner, Shen e Iscoe (1987)]

Aspectos positivos:

- Las etapas están organizadas de un modo lógico. Es decir, si una etapa no puede llevarse a cabo hasta que se hayan tomado ciertas decisiones de más alto nivel, debe esperar hasta que estas decisiones estén tomadas. Así, el diseño espera a los requisitos, el código espera a que el diseño esté terminado, etc.
- Cada etapa incluye cierto proceso de revisión, y se necesita una aceptación del producto antes de que la salida de la etapa pueda usarse. Este ciclo de vida está organizado de modo que se pase el menor número de errores de una etapa a la siguiente.
- El ciclo es iterativo. A pesar de que el flujo básico es de arriba hacia abajo, el ciclo de vida en cascada reconoce que los problemas encontrados en etapas inferiores afectan a las decisiones de las etapas superiores.

MODELO EN V

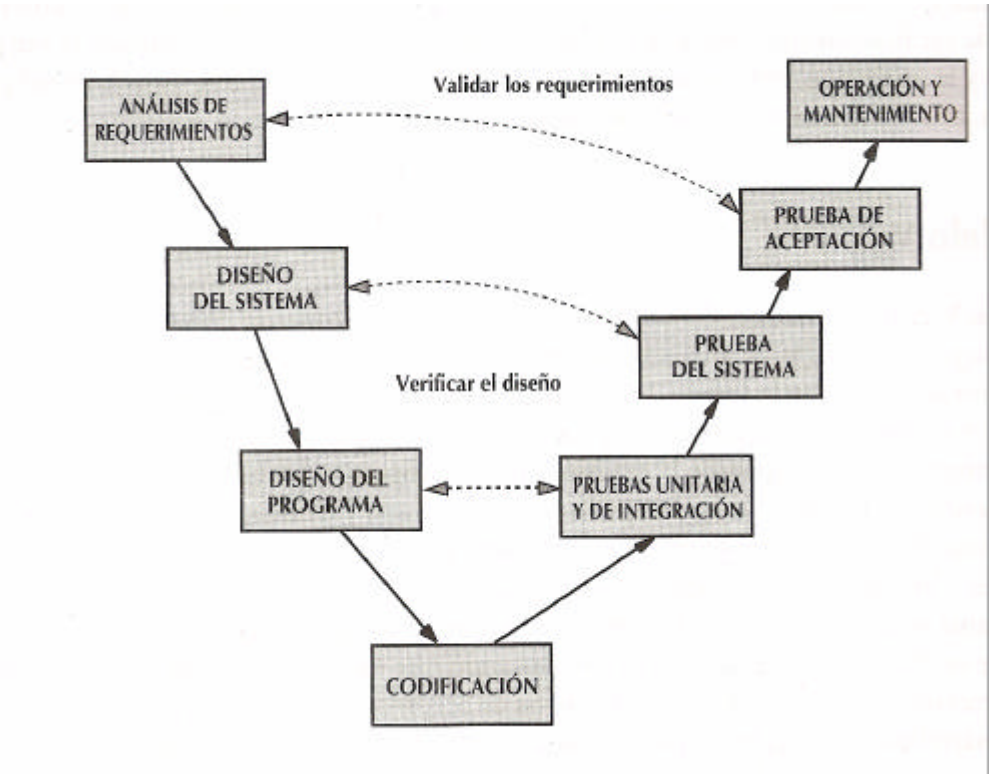

Figura 3 – Modelo en V

Este modelo es una visión alternativa del modelo de ciclo de vida en cascada, y enfatiza en la validación de los productos, y de algún modo en el proceso de composición existente en la construcción de sistemas de software. Este modelo demuestra cómo se relacionan las actividades de prueba con las de análisis y diseño [German Ministry of Defense 1992]

Como se observa en la figura, la codificación forma la punta de la V, con el análisis y el diseño a la izquierda, y la prueba y el mantenimiento a la derecha. La prueba unitaria y de integración se ocupan de la exactitud de los programas. El modelo V sugiere que la prueba unitaria y de integración también sea utilizada para verificar el diseño del programa. Es decir, durante la prueba unitaria y de integración, los codificadores y miembros del equipo de prueba deben asegurar que todos los aspectos del diseño del programa se han implementados correctamente en el código. De igual modo, la prueba del sistema debe verificar el diseño del sistema, asegurando que todos los aspectos del diseño del sistema están correctamente implementados. La prueba de aceptación, que es dirigida por el cliente, en lugar del desarrollador, valida los requerimientos asociando un paso de prueba con cada elemento de la especificación; este tipo de prueba sirve para comprobar si todos los requerimientos se han implementado por completo, antes de que el sistema sea aceptado y pagado.

La vinculación entre los lados derecho e izquierdo del modelo V implica que, si se encuentran problemas durante la verificación y la validación, entonces el lado izquierdo de la V puede ser ejecutado nuevamente para solucionar el problema y mejorar los requerimientos, el diseño y el código antes de retomar los pasos de prueba sobre el lado derecho. En otras palabras, el modelo V hace más explícita parte de la iteración y de rehacer tareas que están ocultas en la representación de cascada. Mientras que el

modelo en cascada centra su foco en los documentos y artefactos producidos, el modelo V lo pone en la actividad y la exactitud.

2. PROTOTIPADO

Suele ocurrir que el cliente no tiene una idea muy detallada de lo que necesita, o que el ingeniero de software no está muy seguro de la viabilidad de la solución que tiene en mente. En estas condiciones, puede ser una mejor aproximación al problema la realización de un prototipo.

Es esta una técnica para proporcionar una versión del sistema software de funcionalidad reducida en las fases iniciales de su desarrollo.

El modelo de desarrollo basado en prototipos tiene como objetivo contrarrestar las limitaciones del modelo en cascada. La idea básica es que el prototipo ayude a comprender los requisitos del usuario. Los prototipos pueden además usarse:

- Para verificar la viabilidad del diseño del sistema.
- Como una herramienta iterativa del desarrollo de software donde el prototipo evoluciona hasta llegar al sistema final.

El problema en el diseño del prototipo es la elección de las funciones que se desea que haga el prototipo, y cuáles son las que no hay que hacer, pues se corre el riesgo de incorporar características secundarias, y dejar de lago alguna característica importante.

Una vez creado el prototipo, se le enseña al cliente para que lo utilice durante un período de tiempo, y a continuación se procede al comienzo del desarrollo a escala real.

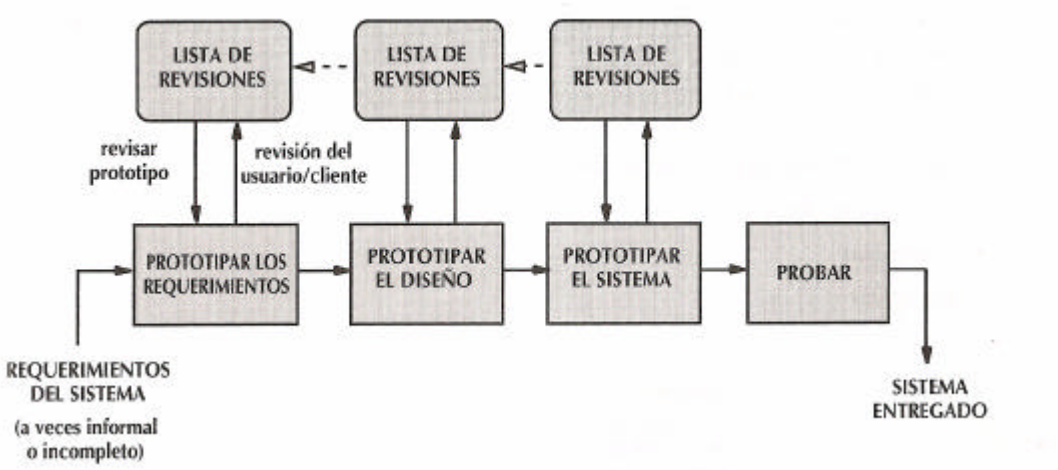

Figura 4– Prototipado

Etapas

1. Análisis preliminar y especificación de requisitos.
2. Diseño, desarrollo e implementación del prototipo.
3. Prueba del prototipo.
4. Refinamiento iterativo del prototipo.
5. Refinamiento de las especificaciones de requisitos.
6. Diseño e implementación del sistema final.

2.1. PROTOTIPO DESECHABLE

Se utiliza para ayudar al cliente a identificar los requisitos de un nuevo sistema. En el prototipo se implantan sólo aquellos aspectos del sistema que se entienden mal o son desconocidos. El usuario, mediante el uso de prototipos, descubrirá esos aspectos o requisitos no captados. Todos los elementos del prototipo serán posteriormente desechados.

2.2. MAQUETA

Aporta al usuario ejemplo visual de entradas y salidas. La diferencia con el anterior es que en los prototipos desechables se utilizan datos reales mientras que en las maquetas son formatos encadenados de entrada y salida con datos simples estáticos.

2.3. EVOLUTIVO

Es un modelo de trabajo del sistema propuesto, fácilmente modificable y ampliable, que aporta a los usuarios una representación física de las partes claves del sistema antes de la implantación. Una vez definidos todos los requisitos, el prototipo evolucionará hacia el sistema final. En los prototipos evolutivos, se implantan aquellos requisitos y necesidades que son claramente entendidos, utilizando diseño y análisis en detalle así como datos reales.

3. MODELOS EVOLUTIVOS

3.1. MODELO INCREMENTAL

A este modelo se lo llama también modelo de ciclo de vida con emisión gradual.

El modelo incremental combina elementos del modelo cascada (aplicados repetidamente) con la filosofía interactiva de la construcción de prototipos.

En el desarrollo incremental, el sistema, tal como está especificado en los documentos de requerimientos, es particionado en subsistemas de acuerdo con su funcionalidad. Las versiones se definen comenzando con un subsistema funcional pequeño y agregando funcionalidad con cada nueva versión. El desarrollo incremental va construyendo gradualmente su funcionalidad completa con cada nueva versión. También proporciona una manera para distribuir periódicamente actualizaciones del mantenimiento del software comercial. Es, por tanto, un modelo popular de la evolución del software usado por firmas comerciales.

Cuando se utiliza un modelo incremental, el primer incremento a menudo es un *producto esencial* (núcleo). Es decir, se afrontan requisitos básicos, pero muchas funciones suplementarias (alguna conocidas, otras no) quedan sin extraer. El cliente utiliza el producto central (o sufre la revisión detallada). Como un resultado de utilización y/o de evaluación, se desarrolla un plan para el incremento siguiente. El plan afronta la modificación del producto central a fin de cumplir mejor las necesidades del cliente y la entrega de funciones, y características adicionales. Este proceso se repite siguiendo la entrega de cada incremento, hasta que se elabore el producto completo.

El modelo de proceso incremental, como la construcción de prototipos y otros enfoques evolutivos, es interactivo por naturaleza. Pero a diferencia de la construcción de

prototipos, el modelo incremental se centra en la entrega de un producto operacional con cada incremento. Al cliente se le entregan funcionalidades cerradas en cada versión.

El desarrollo incremental es particularmente útil cuando la dotación de personal no está disponible para una implementación completa en cuanto a de la fecha límite de gestión que se haya establecido para el proyecto. Los primeros incrementos se pueden implementar con menos personas. Si el producto central es bien recibido, se puede añadir más personal (si se requiere) para implementar el incremento siguiente. Además, los incrementos se pueden planear para gestionar riesgos técnicos. Por ejemplo, un sistema importante podría requerir la disponibilidad de hardware nuevo que esté bajo desarrollo y cuya fecha de entrega no sea cierta. Podría ser posible planificar primeros incrementos de forma que se evite la utilización de este hardware, y así permita entregar la funcionalidad parcial a usuarios finales sin un retraso exagerado.

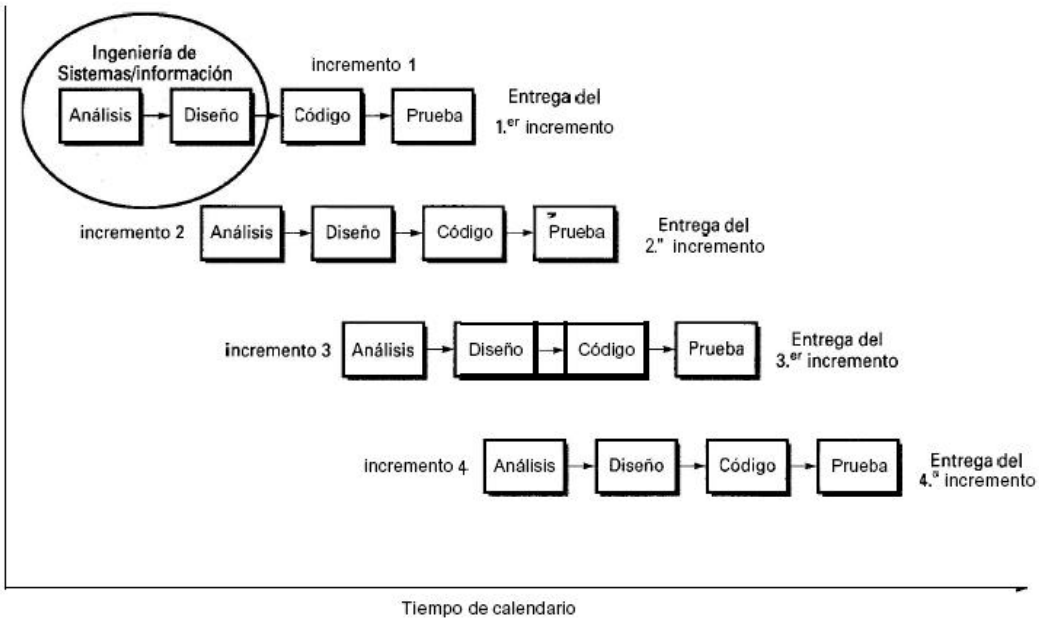

Figura 5 – Modelo Incremental

3.2. MODELO ITERATIVO

A este modelo se lo llama también modelo de refinamiento sucesivo o mejora iterativa.

Las etapas que forman este ciclo de vida son las mismas que el modelo en cascada, y su realización través de un refinamiento y mejora continuos desde las especificaciones de alto nivel del sistema hasta las componentes del código fuente. Es decir, este modelo asume que el producto generado en cada etapa no se produce de manera lineal, del principio al final de la etapa. Por el contrario, predica la generación de los productos de forma iterativa, mediante un proceso de refinamiento. El desarrollo iterativo entrega un sistema completo desde el principio, y luego cambia la funcionalidad de cada subsistema con cada nueva versión.

Debido a la *marcha atrás* permitida en el modelo en cascada, que abre un camino desde una etapa hacia otra anterior, el refinamiento iterativo puede producirse también a nivel global de todas las etapas.

Para comprender la diferencia entre desarrollo incremental y el iterativo analícese un paquete de procesamiento de palabras. Supongamos que el paquete tiene que procesar tres tipos de funcionalidad: creación de texto, organización del texto (es decir, cortar y pegar) y dar formato al texto (como la utilización de diferentes tamaños y estilos de letras). Para construir un sistema como éste utilizando el desarrollo incremental, podemos entregar sólo las funciones de creación en la Versión 1, luego con la Versión 2 las funciones de creación y de organización, y finalmente creación, organización y formato con la Versión 3. Sin embargo, utilizando el desarrollo iterativo debemos proporcionar las formas primitivas de los tres tipos de funcionalidad en la Versión 1. Por ejemplo, se puede crear el texto y luego cortar y pegar, pero las funciones de corte y pegado podrían resultar muy torpes o muy lentas. De este modo la siguiente iteración, Versión 2, tiene esencialmente la misma funcionalidad pero se ha reforzado la calidad: ahora cortar y pegar es fácil y rápido. Cada versión, de alguna manera, es una mejora sobre las precedentes.

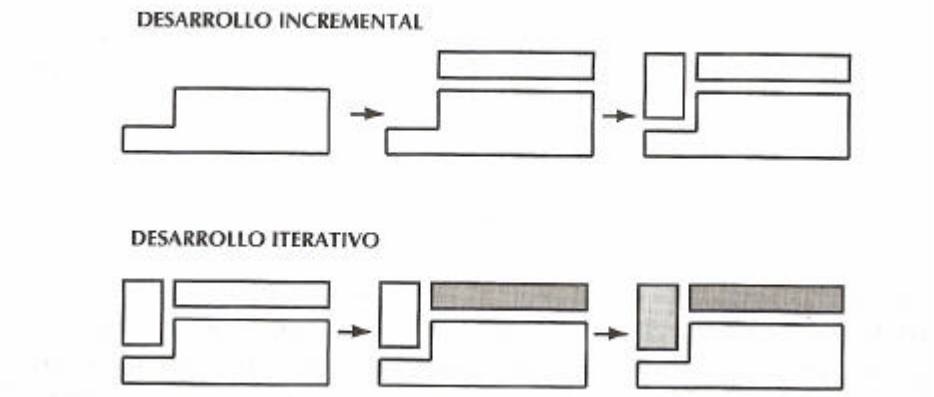

Figura 6 – Los modelos incremental e iterativo

En la realidad muchas organizaciones utilizan una combinación de desarrollo iterativo e incremental. Una nueva versión puede incluir nuevas funcionalidades, pero la funcionalidad que ya existe en la versión actual puede haber sido mejorada. Estas formas de desarrollo por fases son deseables por varias razones:

1. El entrenamiento puede comenzar sobre una versión temprana, aún si se han omitido funcionalidades. El proceso de entrenamiento permite que los desarrolladores observen cómo se ejecutan ciertas funciones, sugiriendo los cambios que las refuercen para versiones posteriores. De esta forma, los desarrolladores pueden dar respuesta a las necesidades de los usuarios.
2. Pueden crearse tempranamente mercados para funcionalidades que nunca antes se habían ofrecido.
3. Las versiones frecuentes permiten que los desarrolladores arreglen problemas no anticipados de manera global y rápida, a medida que son informados desde el sistema operacional.
4. El equipo de desarrollo puede centrarse en diferentes áreas de especialización con las diferentes versiones. Por ejemplo, una versión puede cambiar la modalidad de la interfaz del sistema de la actuada/por/comandos, al modo gráfico (apuntar-y-hacer *click*), utilizando la experiencia de los especialistas en

interfaces de usuario: otra versión puede enfocarse sobre la mejora del rendimiento del sistema.

3.3. MODELO ESPIRAL

El modelo en espiral para el desarrollo de software representa un enfoque dirigido por el riesgo para el análisis y estructuración del proceso software. Fue presentado por primera vez por Böehm [BOE88]. El enfoque incorpora métodos de proceso dirigidos por las especificaciones y por los prototipos. Esto se lleva a cabo representando ciclos de desarrollo iterativos en forma de espiral, denotando los ciclos internos del ciclo de vida, análisis y prototipazo precoz, y los externos, el modelo clásico. La dimensión radial indica los costos del desarrollo acumulativos y la angular, el progreso hecho en cumplimentar cada desarrollo en espiral. El análisis de riesgos, que busca identificar situaciones que pueden causar el fracaso o sobrepasar el presupuesto o plazo, aparece durante cada ciclo de la espiral. En cada ciclo, el análisis de riesgo representa groseramente la misma cantidad de desplazamiento angular, mientras que el volumen desplazado barrido denota crecimiento de los niveles de esfuerzo requeridos para el análisis del riesgo.

Figura 7 – Modelo Espiral

Ventajas

- La primera ventaja del modelo en espiral es que su rango de opciones permiten utilizar los modelo de proceso de construcción de software tradicionales, mientras su orientación al riesgo evita muchas dificultades. De hecho, en situaciones apropiadas, el modelo en espiral proporciona una combinación de los modelos existentes para un proyecto dado.
- Se presta atención a las opciones que permiten la reutilización de software existente.
- Se centra en la eliminación de errores y alternativas poco atractivas.
- No establece una diferenciación entre desarrollo de software y mantenimiento del sistema.
- Proporciona un marco estable para desarrollos integrados hardware-software.

3.3.1. El modelo espiral WINWIN (Victoria&Victoria)

El modelo en espiral tratado en la Sección 3.3. sugiere una actividad del marco de trabajo que aborda la comunicación con el cliente. El objetivo de esta actividad es mostrar los requisitos del cliente. En un contexto ideal, el desarrollador simplemente pregunta al cliente lo que se necesita y el cliente proporciona detalles suficientes para continuar. Desgraciadamente, esto raramente ocurre. En realidad el cliente y el desarrollador entran en un proceso de negociación, donde el cliente puede ser preguntado para sopesar la funcionalidad, rendimiento, y otros productos o características del sistema frente al coste y al tiempo de comercialización.

Las mejores negociaciones se esfuerzan en obtener "victoria-victoria". Esto es, el cliente gana obteniendo el producto o sistema que satisface la mayor parte de sus necesidades y el desarrollador gana trabajando para conseguir presupuestos y lograr una fecha de entrega realista.

El modelo en espiral WINWIN de Boehm [BOE98] define un conjunto de actividades de negociación al principio de cada paso alrededor del espiral. Más que una simple actividad de comunicación con el cliente, se definen las siguientes actividades:

1. Identificación del sistema o subsistemas clave de los "directivos".
2. Determinación de las "condiciones de victoria" de los directivos.
3. Negociación de las condiciones de "victoria" de los directivos para reunirlas en un conjunto de condiciones "victoria-victoria" para todos los afectados incluyendo el equipo del proyecto de software).

Conseguimos completamente estos pasos iniciales se logra un resultado "victoria-victoria", que será el criterio clave para continuar con la definición del sistema y del software. El modelo en espiral WINWIN se ilustra en la figura 8.

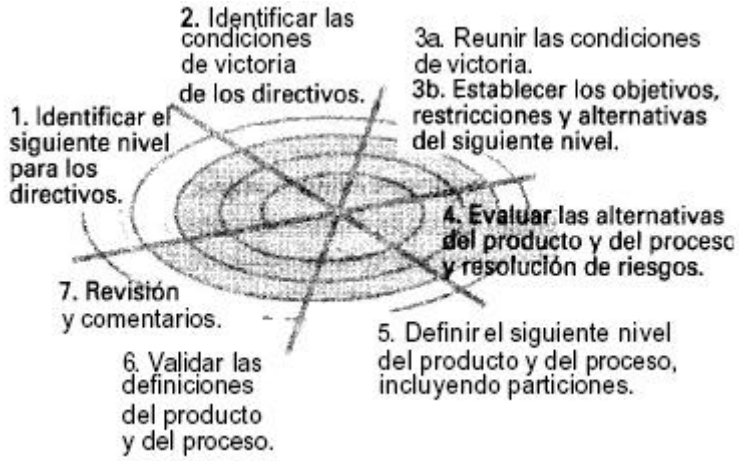

Figura 8 – El Modelo Espiral WINWIN [BOE98].

Además del énfasis realizado en la negociación inicial, el modelo en espiral WINWIN introduce tres hitos en el proceso, llamados ***puntos de*** *fijación* [BOE96], que ayudan a establecer la completitud de un ciclo alrededor de la espiral y proporcionan hitos de decisión antes de continuar el proyecto de software.

En esencia, los puntos de fijación representan tres visiones diferentes del progreso mientras que el proyecto recorre la espiral. El primer punto de fijación, llamado *objetivos del ciclo de vida* (OCV), define un conjunto de objetivos para cada actividad principal de ingeniería del software. Como ejemplo, de una parte de OCV, un conjunto de objetivos asociados a la definición de los requisitos del producto/sistema del nivel más alto. El segundo punto de fijación, llamado *arquitectura del ciclo de vida* (ACV), establece los objetivos que se deben conocer mientras que se define la arquitectura del software y el sistema. Como ejemplo, de una parte de la ACV, el equipo del proyecto de software debe demostrar que ha evaluado la funcionalidad de los componentes del software reutilizables y que ha considerado su impacto en las decisiones de arquitectura. *La capacidad operativa inicial* (COI) es el tercer punto de fijación y representa un conjunto de objetivos asociados a la preparación del software para la instalación/distribución, preparación del lugar previamente a la instalación, y la asistencia precisada de todas las partes que utilizará o mantendrá el software.

4. MODELOS DE CICLO DE VIDA ALTERNATIVOS

4.1. MODELO DRA (Desarrollo Rápido de Aplicaciones)

El Desarrollo Rápido de Aplicaciones es un modelo de proceso del desarrollo del software lineal secuencial que enfatiza un ciclo de desarrollo extremadamente corto. El modelo DRA es una adaptación a alta velocidad del modelo en cascada en el que se logra el desarrollo rápido utilizando un enfoque de construcción basado en componentes. Si se comprenden bien los requisitos y se limita el ámbito del proyecto, el proceso DRA permite al equipo de desarrollo crear un sistema completamente funcional dentro de períodos cortos de tiempo (por ejemplo, de 60 a 90 días)[MAR91]. Cuando se utiliza principalmente para aplicaciones de sistemas de información, el enfoque DRA comprende las siguientes fases [KER94]:

- **Modelado de Gestión.** El flujo de información entre las funciones de gestión se modela de forma que responda a las siguientes preguntas: ¿Qué información conduce el proceso de gestión? ¿A dónde va la información? ¿Quién la genera?
- **Modelado de Datos.** El flujo de información como parte de la fase de modelado de gestión se refina como un conjunto de objetos de datos necesarios para apoyar la empresa. Se definen las características (llamadas atributos) de cada uno de los objetos y las relaciones entre estos objetos.
- **Modelado del Proceso.** Los objetos de datos definidos en la fase de modelado de datos quedan transformados para lograr el flujo de información necesario para implementar una función de gestión. Las descripciones del proceso se crean para añadir, modificar, suprimir, o recuperar un objeto de datos.
- **Generación de aplicaciones.** El DRA asume la utilización de técnicas de cuarta generación. En lugar de crear software con lenguajes de programación de tercera generación, el proceso DRA trabaja para volver a utilizar componentes de programas ya existentes (cuando es posible) o a crear componentes reutilizables (cuando sea necesario). En todos los casos se utilizan herramientas automáticas para facilitar la construcción del software.
- **Pruebas y entrega.** Como el proceso DRA enfatiza la reutilización, ya se han probado muchos de los componentes de los programas. Esto reduce tiempo de pruebas. Sin

embargo, se deben probar todos los componentes nuevos y se deben ejercitar todas las interfases a fondo.

Las limitaciones de tiempo impuestas en un proyecto DRA demandan *ámbito en escalas*. Si una aplicación de gestión puede modularse de forma que permita completarse cada una de las funciones principales en menos de tres meses (utilizando el enfoque descrito anteriormente), es un candidato del DRA. Cada una de las funciones puede ser afrontada por un equipo DRA diferente y ser integradas en un solo conjunto.

Inconvenientes

- Para proyectos grandes aunque por escalas, el DRA requiere recursos humanos suficientes como para crear el número correcto de equipos DRA.
- DRA requiere clientes y desarrolladores comprometidos en las rápidas actividades necesarias para completar un sistema en un marco de tiempo abreviado. Si no hay compromiso, por ninguna de las partes constituyentes, los proyectos DRA fracasarán.

No todos los tipos de aplicaciones son apropiados para DRA. Si un sistema no se puede modularizar adecuadamente, la construcción de los componentes necesarios para DRA será problemático. Si está en juego el alto rendimiento, y se va a conseguir el rendimiento convirtiendo interfases en componentes de sistemas, el enfoque DRA puede que no funcione. DRA no es adecuado cuando los riesgos técnicos son altos. Esto ocurre cuando una nueva aplicación hace uso de tecnologías nuevas, o cuando el nuevo software requiere un alto grado de interoperatividad con programas de computadora ya existentes.

DRA enfatiza el desarrollo de componentes de programas reutilizables. La reutilización es la piedra angular de las tecnologías de objetos.

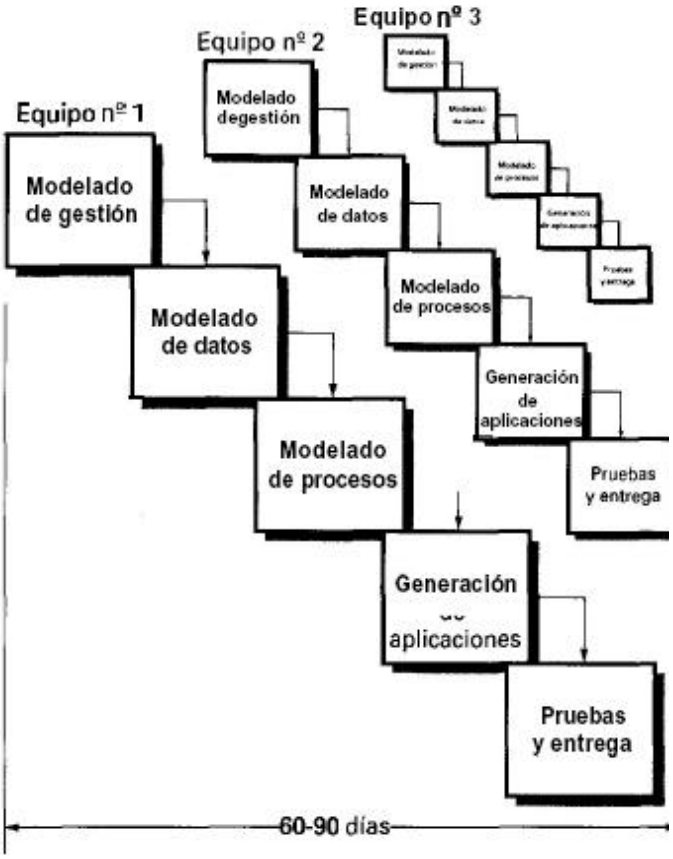

Figura 9 – Modelo DRA

4.2. ESTÁNDARES MILITARES Y PRÁCTICAS INDUSTRIALES

Las empresas industriales adoptan con frecuencia alguna variación del modelo clásico como base de la práctica del desarrollo software. Muchos suministradores de la administración americana organizan sus actividades de acuerdo con los modelos de ciclo de vida del estándar militar, tal como el que se engloba en la norma MIL-STD-2167 de 1978. Tales estándares subrayan no sólo alguna variación de las actividades del ciclo de vida clásico, sino que también contienen los documentos que deben entregarse a los clientes que necesitan sistemas software o mecanismos complejos con sistemas software embebidos. Estos estándares se intenta que sean también compatibles con la garantía de calidad del software, la gestión de configuraciones y verificación y validación independiente de servicios en un proyecto de desarrollo con más de un contratista. Estos modelos ponen especial énfasis en la definición de productos entregables, revisiones, hitos y técnicas requeridas en cada caso.

4.2.1. Introducción a la norma MIL-STD-2167

Éste es el estándar utilizado por las Fuerzas Armadas de EE.UU. En realidad lo utilizan como normativa que deben cumplir los contratistas que se dedican a hacer productos software para ellos. En este estándar se contemplaba el desarrollo de productos integrados que incluyen hardware y software, y se establecían ciclos de vida paralelos para ambos. En la última versión, se han separado los conceptos del

software y hardware de manera que, en la actualidad, este estándar no afecta más que al software.

En el estándar se considera que existe un sistema que se divide en varios CSCI (*Computer Software Configuration Item*). Cada CSCI se divide en varios CSC (*Computer Software Component*). Y cada CSC se divide a su vez en CSU (*Configuration Software Units*). Esta división es importante ya que afecta directamente a la división en etapas del ciclo de vida.

Etapas

- **Análisis de requisitos del sistema global** (sistema físico más software embebido).
- **Diseño del sistema.**
- **Análisis de requisitos del software.**
- **Diseño preliminar.**
- **Diseño detallado.**
- **Codificación y verificación de CSUs.**
- **Integración y verificación de CSUs.**
- **Prueba CSCIs.**
- **Integración y prueba del sistema.**

En cada etapa del ciclo de vida, se especifican los documentos que se tienen que generar, al igual que las revisiones que debe sufrir el producto. Este estándar es un poco excesivo en cuanto a documentación requerida, y además pide más de un documento que tiene información redundante.

4.2.2. Introducción a la norma ESA PSS-05-0

Ésta es la norma utilizada por la Agencia Espacial Europea (ESA en inglés) para sus desarrollos software. Al igual que el de las Fuerzas Armadas Americanas, este documento está orientado principalmente a las empresas que desarrollan software bajo contrato, ya que la Agencia desarrolla poco software por sí misma. Estos estándares son bastante generales, y cada proyectos de gran envergadura suele desarrollarlos más, haciendo los estándares propios del proyecto. En caso de conflicto, se suele establecer una jerarquía, en la que prima el estándar más general.

En este estándar, se consideran todos los aspectos de un proyecto de software. Además del propio desarrollo y su normativa, se contemplan los aspectos de:

- Gestión del proyecto.
- Gestión de configuración.
- Control y garantía de calidad.

Etapas

- **Definición de requisitos del usuario.**
- **Definición de requisitos del software.**
- **Diseño de la arquitectura.**

- **Diseño detallado y producción del software.**
- **Transferencia de la tecnología al usuario.**
- **Operación y mantenimiento.**

La etapa que aparece nueva, respecto al ciclo de vida clásico, es la de la transferencia. En esta fase la Agencia, viendo los resultados de las pruebas, da su aceptación provisional al software, y se procede a la instalación en la máquina objetivo, a la formación de los usuarios, etc. Esta fase tiene sentido, en tanto y en cuanto, el software se suele desarrollar en países distintos de aquel en que se va a instalar posteriormente, y por lo tanto, la fase de instalación precisa de desplazamiento de personal, y otras peculiaridades.

En este estándar se contemplan para cada fase los siguientes factores:

- Entradas
- Actividades
- Salidas

Además, y para cada uno de los tres aspectos mencionados: gestión del proyecto y de las configuraciones y control de la calidad, se describen los planes necesarios para su ejecución, así como los documentos e hitos asociados.

La ventaja fundamental de este estándar es que es muy sencillo y fácil de comprender, con lo que se puede tomar como punto de referencia para desarrollos propios en cualquier otro entorno.

4.3. MODELO DE ENSAMBLAJE DE COMPONENTES

Las tecnologías de objetos proporcionan el marco de trabajo técnico para un modelo de proceso basado en componentes para la ingeniería del software. El paradigma de orientación a objetos enfatiza creación de clases que encapsulan tanto los datos como los algoritmos que se utilizan para manipular los datos. Si se diseñan y se implementan adecuadamente, las clases orientadas a objetos son reutilizables por las diferentes aplicaciones y arquitecturas de sistemas basados en computadoras.

El modelo de ensamblaje de componentes incorpora muchas de las características del modelo en espiral. Es evolutivo por naturaleza [NIE92], y exige un enfoque interactivo para la creación del software. Sin embargo, el modelo ensamblador de componentes configura aplicaciones desde componentes preparados de software (algunas veces llamados *clases*).

La actividad de la ingeniería comienza con la identificación de clases candidatas. Esto se lleva a cabo examinando los datos que se van a manejar por parte de la aplicación y el algoritmo que se va a aplicar para conseguir el tratamiento. Los datos y los algoritmos correspondientes se empaquetan en una clase.

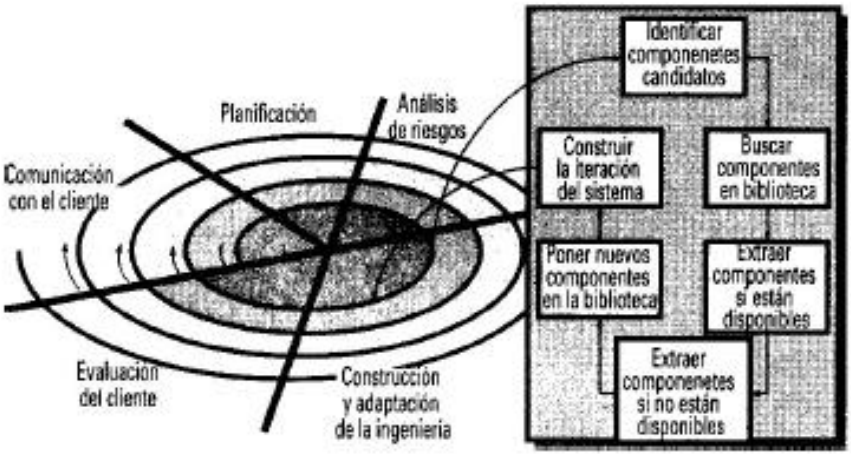

Figura 10 – Modelo de Ensamblaje de Componentes

Las clases (llamadas componentes en la figura anterior) creadas en los proyectos de ingeniería del software anteriores se almacenan en una *biblioteca de clases* o *propósito*. Una vez identificadas las clases candidatas, la biblioteca de clases se examina para determinar si estas clases ya existen. En caso de que así fuera, se extraen de la biblioteca y se vuelven a utilizar. Si una clase candidata no reside en la biblioteca, se aplican métodos orientados a objetos. Se compone así la primera interacción de la aplicación a construirse, mediante las clases extraídas de la biblioteca y las clases nuevas construidas para cumplir las necesidades únicas de la aplicación. El flujo del proceso vuelve a la espiral y volverá a introducir por último la iteración ensambladora de componentes a través de la actividad de la ingeniería.

El modelo ensamblador de componentes lleva a la reutilización del software, y la reutilización proporciona beneficios a los ingenieros del software en cuanto a la utilización de recursos.

5. BIBLIOGRAFIA

- *Sommerville, Ian*

  Ingenierìa del Software; Pearson Education, México 2002. ISBN: 970-26-0206-8

  Capítulo 3 – Procesos de Software. Páginas 44 a 55.

- *Pfleeger, Shari*

  Ingenierìa del Software, teoría y práctica; Pearson Education, Buenos Aires 2002. ISBN: 987-9460-71-5

  Capítulo 2 – Modelado del proceso y del ciclo de vida. Páginas 54 a 66.

- *Pressman, Roger*

  Ingenierìa del Software, un enfoque práctico; Mc Graw Hill, 1998. ISBN: 0-07-052182

  Capítulo 2 – El proceso. Páginas 21 a 34.

- *Jurista, Natalia*

  Modelos de Proceso Software; Universidad Politécnica de Madrid, Madrid 2000.

  Páginas 8 a 23.